Improving Floating Point Compression through Binary Masks

Leonardo A. Bautista Gomez
Argonne National Laboratory

Franck Cappello
Argonne National Laboratory

***Abstract*—Modern scientific technology such as particle accelerators, telescopes and supercomputers are producing extremely large amounts of data. That scientific data needs to be processed using systems with high computational capabilities such as supercomputers. Given that the scientific data is increasing in size at an exponential rate, storing and accessing the data is becoming expensive in both, time and space. Most of this scientific data is stored using floating point representation. Scientific applications executed in supercomputers spend a large amount of CPU cycles reading and writing floating point values, making data compression techniques an interesting way to increase computing efficiency. Given the accuracy requirements of scientific computing, we only focus on lossless data compression. In this paper we propose a masking technique that partially decreases the entropy of scientific datasets allowing for better compression ratio and higher throughput. We evaluate several data partitioning techniques for selective compression and compare these schemes with several existing compression strategies. Our approach shows up to 15% improvement in compression ratio while reducing the time spent in compression, to only a half of the original compression time in some cases.**

I. Introduction

Nowadays, the amount of data produced worldwide in a year is larger than the amount of data produced by human kind during centuries. A large amount of it corresponds to scientific data generated in scientific facilities, using advanced technology with large amount of sensors. The scientific data is then analyzed and processed, which also generates more data. For instance, scientific applications running in large supercomputers generate large amounts of data that need to be stored on persistent storage for later post-processing. In some cases that data corresponds to an application state that is saved and used to restart if a failure occurs (i.e. checkpointing), it can be used to analyze the behavior of the application (i.e. profiling) or to visualize the physical phenomena modeled by the application (i.e. visualization). Independently of the purpose, saving it on reliable storage is becoming prohibitely time consuming as the size of the data is increasing much faster than the I/O system of supercomputers [1].

In order to decrease the time to write scientific data to the Parallel File System (PFS), it is possible to reduce the size of the data by using data compression. Data compression has been widely used in a wide range of areas and it is becoming critically important for modern scientific research as the dataset sizes are increasing exponentially. Data compression can come in two flavors, lossy data compression and lossless data compression. Lossy compression is used in applications that tolerate data-loss without altering its performance (e.g. audio compression). Scientific computing however, might suffer of large inaccuracies or even failure to correctly conclude the execution, given as less as one single bit alteration. Therefore, lossless compression is usually the preferred choice while compressing scientific data.

In addition to the overwhelming amount of data, another challenge of dealing with scientific data is that most of it uses the IEEE floating point data representation [2]. In High Performance Computing (HPC), a high level of precision is often required to guarantee scientifically meaningful results. Thus, most of data used in HPC applications uses single (32 bits) or double precision (64 bits) floating point representation. Each value is composed of three parts, the sign, the exponent and the mantissa. While the sign and the exponent of floating point values in a data set might show some regularity, the largest part of the floating point values (i.e. the mantissa) is highly irregular, making it hardly compressible.

While there is wide literature on data compression, the most popular compression techniques are general purpose compressors that have as first goal to perform decently in most cases rather than focusing in particular types of datasets. Then, there is a range of compressors that target some particular cases, such as digital pictures, audio or text files. Unfortunately, there has been only few (yet important) work on compressing scientific data. In this work we aim to study the limits of lossless data compression for floating point datasets. The contributions of this work can be summarized as follows:

- We propose a masking technique that partially decreases the entropy level of HPC datasets leading to datasets with higher compressibility levels.
- We implemented a fast compression algorithm that focuses in floating point data using binary masks.
- We evaluated our masking technique with real scientific datasets and demonstrate an improvement on compression ratio up to 15% and up to 2x in compression throughput.

The rest of this paper is organized as follows: Section II explain the motivations for this research, Section III presents our masking technique introduced in this work. Section IV shows our evaluation, Section V overviews the related work and finally Section VI concludes this work.

II. Motivations

The research work presented in this paper has as primary goal to decrease the cost of storing, accessing and moving

large amounts of scientific data on future large scale computing facilities where such data is usually processed and utilised. Then, two main challenges arise: the size and the irregularity of the data. In this section we present why these two characteristics are so challenging while working with scientific data.

A. The Big Data Era

Modern science is generating large amounts of data that needs to be analyzed and used for different purposes. For instance, the Large Hadron Collider (LHC) in Switzerland generates about 25 Petabytes of data every year. This data is distributed among 170 computing facilities thanks to the LHC worldwide Computing Grid infrastructure [3], [4]. Transmitting over the network such large amounts of data from one computing center to another can take a large amount of time, limiting scientific collaboration. Therefore, compressing scientific data is critical for the efficient use of such collaborative networks. In addition to the already existing scientific challenges, the future of scientific discoveries seems to be closely related to the capacity to get, store, access and process larges amount of data. For instance, there are plans to build a telescope that is expected to generate more data than the whole internet in one single day [5]. Storing scientific data for projects of such magnitude will require millions of storage devices making important any gain in space.

Large supercomputers process scientific data and perform complex computation tasks on that data. Unfortunately the size of scientific data is growing exponentially and the bandwidth to access that data is only increasing linearly. One example of this mismatch between the size of the data and the bandwidth to write that data in a reliable storage is the case of HPC applications checkpointing. Indeed, when applications write their state into the PFS, they lose a large amount of CPU cycles waiting to finish to write the checkpoint files. More generally, the discrepancy between computing capabilities and data movement cost is increasing so quickly that the HPC community has arrived to the conclusion that in the future flops will be *free* while data movements will be costly in both, time and energy. This shows the importance of scientific data compression for future large computing systems.

Precision	Sign	Exponent	Mantissa
Single	1	8	23
Double	1	11	52

TABLE I: Floating point representation

B. The Scientific Data

Most of the scientific data used in HPC is represented in IEEE floating point representation, in either single or double precision. Table I shows the bit distribution of floating point values for both precisions (32 and 64 bits). One single HPC application could have tens of different variables, each one with a particular physical meaning. From the data compression perspective this data diversity is already a first challenge. Indeed, if the data is not written in an organized fashion (i.e. all values with the same physical meaning together), it could be very difficult for any compression algorithm to find patterns that would be obvious in other circumstances.

Given that values of variables with the same meaning are somehow *similar* (i.e. same orders of magnitude), grouping them increases the local regularity of the data. Also, many phenomena will show a gradual variation across the dataset. While such gradual variation could be more or less drastic depending on the given scenario, in many cases the average variation between neighbor elements is relatively *low*, which could have a significant impact on the regularity of the datasets and therefore their compressibility.

Fig. 1: Floating Point Data: Each column is the binary representation of a 64 bits floating point value. A black square means 0 and a white square means 1

Unfortunately, even while scientific datasets are written keeping structures in a good order, floating point data usually involves a high level of entropy (i.e. high irregularity), in particular in the less significant bits of the mantissa. Figure 1 shows the binary representation of the 64 bits floating point data of a numerical simulation of the brain. Each value is expressed in a column of 64 small squares, using white color for a 1 and black for a 0. As we can see the first bits of the values present a high regularity (upper part of the figure) while the less significant bits of the mantissa show high irregularity (lower part of the figure). While some datasets seem hard to compress, restructuring or partitioning strategies might increase the achievable compression ratio.

III. Masking data

The first technique that we propose in this work is based on the idea of altering the datasets to bring them as close as possible to the point of *Zero entropy*. A *Zero entropy* dataset can be described as a dataset where all the bits are equal (i.e. either all 1 or all 0) [6]. Assuming we could find a bijection that could transform a scientific dataset into a *Zero entropy* dataset then we could compress the resulting dataset to an negligible amount of information: the size of the dataset. One could imagine that it is impossible to transform every possible dataset into a *Zero entropy* dataset and then recover the original data using a simple bijection. But it is possible if for every dataset we use a different bijection. In other words, if it was

possible to automatically find for each dataset a bijection that transforms the dataset into a *Zero entropy* dataset, we could compress the resulting data and just store the compressed data together with the bijection.

A. The silly case

Now that the idea of reducing the variability of the scientific datasets has been introduced we can focus on the automatic generation of such bijections. The main concept relies on applying *masks* to the data so that each value is transformed into a *Zero entropy* value (e.g. 1111...111), and store the masks (i.e. the bijection) to be able to recover the original data. Figure 2 shows an example in which a mask is applied to a value using XOR encoding. If we assume that one mask is applied to each value, then the number of masks will be equal to the number of values. Given that the size of the masks is the same size of each value, then we will end up with a bijection that contains the same amount of information as the original dataset: hence we call this case the *silly case*. Although this case does not bring any advantage in terms of data compression, the idea can still be exploited if we find a way to reduce the amount of information in the bijection.

	S	Exponent								Mantissa											
Value:	1	1	0	0	1	0	0	0	1	0	1	0	1	0	1	...	1	0	1
Mask:	0	0	1	1	0	1	1	1	0	1	0	1	0	1	0	...	0	1	0
Masked:	1	1	1	1	1	1	1	1	1	1	1	1		1	1	...	1	1	1

Fig. 2: Masking Floating Point Values

One option to reduce the size of the bijection is to reduce the number of masks. Thus we could imagine a configuration in which all the values of a dataset use the same mask. For this purpose we could study the percentage of ones and zeros in each column of the vector and apply a mask that will increase the overall percentage of ones. For instance, if 100% of the bits in the first column are ones and 100% of bits in the next column are zeros, a mask starting by 01... would keep the first column as it is and at the same time change all the values of the second column, giving as result a dataset where 100% of both, the first and the second column are ones. Now if we assume that the first 16 columns (2 bytes) are completely regular then by applying the same mask to all values we could get as result a dataset where all the first 16 columns represent a partial *Zero entropy* dataset. Unfortunately, performing statistics over the entire dataset is unrealistic given the large size of modern scientific data. Moreover, such technique will only increase the similarities between bit columns but it will not change the overall regularity of each column.

B. Exploiting neighbor proximity

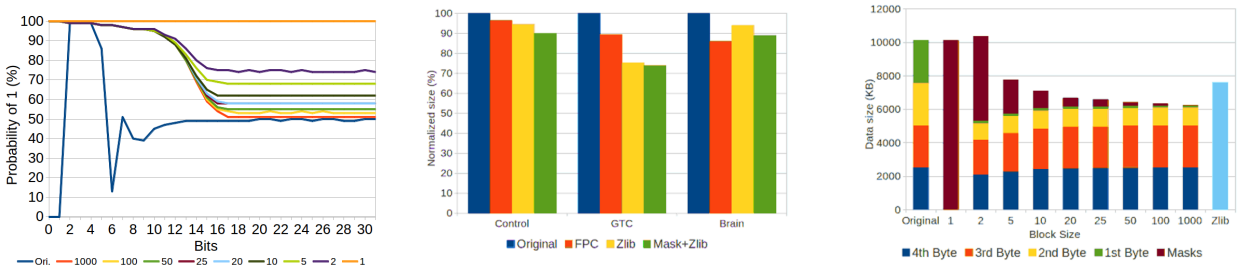
In order to increase the regularity of a dataset while keeping a low overhead due to masks storing, we propose to apply one mask to a block of data. The idea being that most HPC applications have datasets in which neighbor elements have close values. For instance, if we apply the same mask to a block of two floating point values, the mask overhead in relation to the original dataset size will be 50%. Thus, the same could be done with blocks of 10 values for a mask overhead of 10%, etc. To generate a mask for a block of values we first do a statistical study for each bit column and then generate a *signature value* for the block in which each bit is decided based on the most frequent value for the given bit column. Then we generate a mask for the *signature value* in the same way that we do it for one single value. The mask for the next block might be different but the regularity of the dataset increases by applying different masks to different blocks of data. Blocks of small numbers of values imply a larger number of masks (hence larger mask overhead) but it also increases the regularity of the dataset. On the other hand, applying the same mask to a large number of values decreases the probability of successful irregularity reduction but it also involves only a negligible overhead for storing the masks. This trade-off between irregularity reduction and mask overhead will be studied in more detail in Section IV.

C. Masked data compression

After the masks have been applied to the original data, the masked data is compressed using any classic data compressor such as *zlib* [7]. Our first approach is to compress the data after it has been masked without using any partitioning technique. This is the most simple case given that not extra work is required after masking. It is important to notice that the classic compression algorithms also process the data in blocks. Thus, we decompose the large compression blocks into smaller blocks that are then passed to the masking engine.

D. Byte-level column-wise compression

In a second approach we apply byte-level column-wise serialization before the compression process. To this purpose, we developed a module that partition the data in 4 or 8 datasets (single or double precision accordingly), each one of them corresponding to a byte column of the floating point data. While the statistical analysis performed during the masking process could give us hints on whether a particular byte-column could avoid compression to increase throughput and efficiency, we do perform the compression of all the columns in order to analyze if the masking strategy can alleviate the situation on the lower bits of the mantissa. This scenario is similar to the ISOBAR-compress technique [8], with the difference that in this work we study the impact of applying masks to the data before compression. One of the important aspects to study while partitioning the data in columns is to try to understand the impact of the masking strategy for different byte positions. Assuming that only some part of the floating point values benefits from the masking technique, applying full size masks is a waste of space. In such scenarios, applying mask to only a subset of bytes could be sufficient to achieve the same compression ratio.



(a) Per Bit Value Distribution (b) Nomalized Data Size (c) Byte Column-wise Compression Analysis

Fig. 3: Comparison between masked-data compression vs. other compression techniques. The first and third experiments use a 10MB particles dataset of the GTC application (Single precision). The second experiment is based on a human brain simulation's 136MB dataset in double precision and a control vector from a weather simulation.

IV. Evaluation

In our evaluation we first study the impact of our masking technique on the regularity of the datasets and we compare it to other compression schemes. Then, we study the impact of the byte-level column-wise encoding strategy and we show the efficiency gained with partial masking, in both compression ratio and compression throughput and we show the stability of our compression technique. We test our proposed techniques with multiple datasets including the numerical simulation of human brain during impact (velocity field), particle data for a 3D Gyrokinetic Toroidal Code that makes part of the NERSC benchmarks [9], a control vector used for the assimilation of weather data, data from several NASA Parallel Benchmarks (NPB) [10], simulated plasma temperature and many others. The datasets have between tens and hundreds MBs of floating point data, in single or double precision.

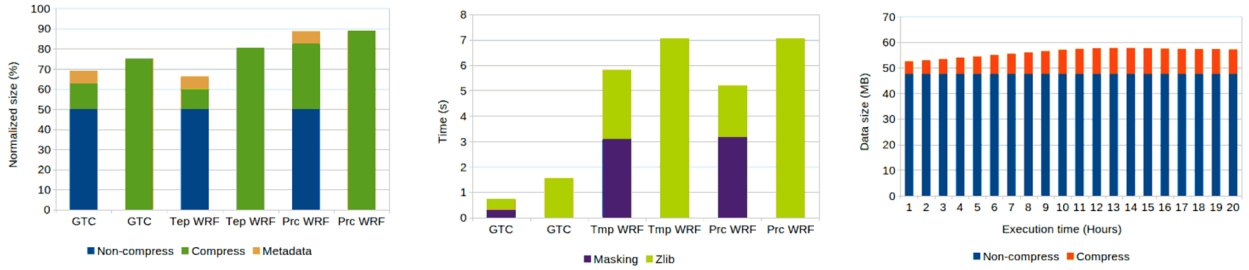
A. Masking study

In this section we study the impact of our masking technique. We start by applying masks to blocks of 1000 float values and then decrease the size of the blocks to 1 single floating point value. We perform a bit-column analysis of the dataset in which we measure the probability of a bit being equal to 1 depending on its position. The same analysis is performed on the original and masked data for multiple block sizes. The results are presented in Figure 3a. As we observe, the original dataset is very regular in a per-column basis, for the first few bits. Then, the regularity of the data decreases exponentially for the mantissa bits. Overall, we notice that the masking technique does improve the regularity of the data. It is interesting to see that the smaller the block size, the higher the regularity of the data; up to some point where the dataset is transformed into a *Zero entropy* dataset using a masking block size of one single floating point value. This *silly case*, is obviously not interesting for the compression aspect of this research, but it does help us to assert that our masking algorithm is working correctly.

Then, we compare our masked-data compression scheme with other compression techniques for several applications. We measure the size of the original data and the size of the compressed data using three techniques. The first technique is the Floating Point Compression (FPC) [11] library. The second one is the Zlib general purpose compression library [7] and the last one is using Zlib to the masked data. Figure 3b shows the results of our compression evaluation. For clarity we have normalized the results to the original size of each dataset. We notice that the masked data Zlib compression is systematically better than the plain Zlib compression, although the difference between both depends on the dataset. We also notice that FPC does not always performs better than Zlib, but in some cases it performs better than Zlib with and without masking. We then perform a more detailed analysis by measuring which bytes are more impacted by the our masking technique and how that is reflected on the final compression ratio.

B. Column-wise compression study

In order to study how our masking technique impacts the different byte columns of the floating point data, we implement an experiment in which the data is masked and reorganized in columns of bytes before compression. For instance, a dataset of single precision floating point values will be distributed in four subsets of data, where the x^{th} subset holds the x^{th} byte of each float. This redistribution is done after the original data is masked. Then, we compress each subset of data and store them together with the set of masks and all metadata required. We measure how much space is necessary to store each compressed subset and the set of masks. The results are plot in Figure 3c. The first case at the left shows the space required to store each byte-column on the original data before masking and compression. Then, we can see the *silly case* of one mask per float. The space required to store the masks is equal to the original size of the data, as expected. Then, we vary the block size from 2 floats up to 1000 floats. we observe that the block size does impact the compression ratio for all bytes, but at different levels. For instance, the first byte shows an impressive compression ratio for any block size.



(a) Byte Column-wise Compression Comparison (b) Compression time comparison (c) Compression ratio stability

Fig. 4: Compression decomposition, throughput and stability study. This experiments use data from the GTC application and different variables (temperature and precipitation) from a numerical simulation of the Hurricane Isabela done with the Weather Research and Forecasting (WRF) model. More variables were analyzed giving similar results.

C. Partial masking study

This evaluation shows that the masking techniques increases the regularity of some byte columns. However, it is also clear that the masks are not improving the regularity of the mantissa bits. Storing masks for bytes that do not are getting any benefit from it, is a waste of storage. Thus, we improved our software, in such way that it only stores the masks of those columns that are getting improvements. This allows us to have an even finer block decomposition because the overhead related to the masks is decreasing to half or a quarter of the original overhead. In addition, we also avoid compressing high entropy byte columns because they show a low compression ratio and it requires an important amount of time. This idea is similar to the ISOBAR [8] library but we extend that work with our masking technique, to increase the number of columns that present compressible patterns. By compressing only a part of the data we increase the compression speed and increase the compression ratio of the compressible part of the data. We perform an experiment in which we measure the amount of compressed data, non-compressed data and metadata (including masks) that our approach generates. Figure 4a shows the results for three different HPC datasets, the last two correspond to checkpoint data generated while performing a simulation of the Hurricane Isabela using WRF [12]. As we can see, masking and compressing only part of the data shows important gains in compression ratio in comparison to compressing the plain dataset. Our complete technique shows up to 15% of improvement in the compression ratio.

We also measure the masking and compression speed and compare it with a plain compression of the whole dataset. Our results are presented in Figure 4b. As we can observe, the masking and decomposition process takes a significant amount of time, but this is tolerable because it greatly reduces the following compressing time. In fact, in some cases, masking and compressing takes half of the time that it takes to compress the whole dataset. In all cases, the compression speed of our approach is largely higher than compressing the plain data. Thus, not only our approach present better compression ratio but it also offers dramatical speedup. It is also important to notice, that at this point not much effort was done to optimize our masking scheme, but an statistical approach in which only a subset of the values is analyzed could present similar compression ratio for even higher masking/compression speed.

D. Compression ratio stability

Although we have performed a complete evaluation with multiple different HPC datasets, one could argue that high compression ratios are achieved because the initial conditions of HPC applications present a high degree of homogeneity. To show that this is not the case for our results, we perform an experiment in which we compress the checkpoint data of an execution that lasted over 20 hours. Checkpoints are taken every hour. The execution corresponds to the simulation of the Hurricane Isabela using WRF. In this test we compress the temperature variable which corresponds to a 3D structure. Here we focus only on the compressible data versus non-compressible data. The values are stored in single precision (32 bits) and the masks are 16 bits long (i.e. decomposition by half). Figure 4c shows the results of this experiment. As we can see, the compressed data is only a fraction of its original size, the compression ratio is dramatically high for the compressible bytes. Although, the compressed size does increase slightly during the first hours of the execution, it is clear that even after 20 hours of execution our approach stabilises and still guarantees a very high compression ratio.

V. Related work

There is a large literature on data compression but only a small percentage of it focuses on floating point compression which is critical for big data science. Isenburg et al. [13] propose a lossless compression technique for floating point coordinates for 3D structures. The authors propose to predict the following floating point value and then compress the residual between the prediction and the actual value, achieving impressive results for the compression of meshes and other geometric structures. It is important to notice that 3D structures are likely to have more predictive patterns in comparison with scientific data that might contain a substantial amount of noise.

Lindstrom et al. [11] propose a more general approach that leverages the results of the above-mentioned method. While the prediction of floating point data gives as result a value very close to the actual value, the high precision of floating point data limits the benefits of such prediction. Better results are achieved in this work thanks to an improved entropy coder. While this work achieves good results, in both compression ratio and compression throughput, it does not study which part of the floating point data is responsible for the results and if it is really necessary to process the whole dataset.

One of the most relevant works is the ISOBAR-compress library [8] that partitions the floating point data in compressible and non-compressible data using statistical analysis. This technique achieves state-of-the-art compression ratio and compression throughput given that only a subsection of the whole dataset is passed to the compressor engine. This technique analysis the regularity (i.e. entropy) of each byte column and decides whether is worth or not to try to compress the given column. While this is close to part of the work presented in this paper, an important difference is our masking strategy that shows significant improvements in terms of compression ratio. Also, we perform a more detailed study in which we demonstrate the stability of the compression ratio.

Floating point decomposition has also been exploited for visualization purposes [14]. The idea is to read floating point data with a partial precision, so that the amount of data read is much lower than the original data size. However, this approach does not reduce the amount of data written to the PFS. Another relevant work is PRIMACY [15]. PRIMACY is a preconditioner that enhances the regularity of floating point scientific datasets before they are passed to regular compressors. Their analysis of the datasets is similar to the one used in this work. However, the usage of binary masks as a way to increase the regularity of the data is not presented in that work. We believe this work could complement our work and biseversa, in the sense that by mixing both *preconditioners* we could achieve even higher data regularity.

VI. Conclusion

In this research work we have proposed binary masks as a way to increase the regularity of HPC datasets. Then, once the masks have been applied, we can analyze which parts of the dataset are worth compressing and are passed to the compressor. This approach has shown up to 15% of improvement in compression ratio, while the compression can take only half of the time in some cases. This technique reduces the time and space required to compress floating point data. In the future, we plan to study some statistical approaches that could optimize further our masking scheme. Also, we would like to explore multi-process compression [16] as a way to exploit similarities between processes.

VII. Acknowledgement

This work was supported by the U. S. Department of Energy, Office of Science, under Contract No. DE-AC02-06CH11357.

References

[1] R. A. Oldfield, S. Arunagiri, P. J. Teller, S. Seelam, M. R. Varela, R. Riesen, and P. C. Roth, "Modeling the Impact of Checkpoints on Next-Generation Systems," in *MSST '07: Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 30–46.

[2] IEEE, "IEEE Standard for Floating-Point Arithmetic," http://standards.ieee.org/findstds/standard/754-2008.html.

[3] WLCG, "What is the Worldwide LHC Computing Grid," http://lcg-archive.web.cern.ch/lcg-archive/public/overview.htm.

[4] ——, "Worldwide LHC Computing Grid," http://wlcg.web.cern.ch/.

[5] CNN, "A telescope that generates more data than the whole internet," http://www.cnn.com/2012/04/02/tech/innovation/ibm-big-data-telescope.

[6] C. E. Shannon and W. Weaver, "A mathematical theory of communication," 1948.

[7] J.-l. Gailly and M. Adler, "Zlib compression library," 2004.

[8] E. Schendel, Y. Jin, N. Shah, J. Chen, C. S. Chang, S.-H. Ku, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. Samatova, "Isobar preconditioner for effective and high-throughput lossless data compression," in *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, 2012, pp. 138–149.

[9] N. Wichmann, M. Adams, and S. Ethier, "New advances in the gyrokinetic toroidal code and their impact on performance on the cray xt series," *Cray Users Group*, 2007.

[10] D. H. Bailey, E. Barszcz, L. Dagum, and H. D. Simon, "Nas parallel benchmark results," *Parallel & Distributed Technology: Systems & Applications, IEEE*, vol. 1, no. 1, pp. 43–51, 1993.

[11] P. Lindstrom and M. Isenburg, "Fast and efficient compression of floating-point data," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 12, no. 5, pp. 1245–1250, 2006.

[12] W. C. Skamarock and J. B. Klemp, "A time-split nonhydrostatic atmospheric model for weather research and forecasting applications," *Journal of Computational Physics*, vol. 227, no. 7, pp. 3465 – 3485, 2008, ¡ce:title¿Predicting weather, climate and extreme events¡/ce:title¿. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0021999107000459

[13] M. Isenburg, P. Lindstrom, and J. Snoeyink, "Lossless compression of predicted floating-point geometry," *Comput. Aided Des.*, vol. 37, no. 8, pp. 869–877, Jul. 2005. [Online]. Available: http://dx.doi.org/10.1016/j.cad.2004.09.015

[14] J. Jenkins, E. Schendel, S. Lakshminarasimhan, D. Boyuka, T. Rogers, S. Ethier, R. Ross, S. Klasky, and N. Samatova, "Byte-precision level of detail processing for variable precision analytics," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, 2012, pp. 1–11.

[15] N. Shah, E. R. Schendel, S. Lakshminarasimhan, S. V. Pendse, T. Rogers, and N. F. Samatova, "Improving i/o throughput with primacy: Preconditioning id-mapper for compressing incompressibility," in *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*. IEEE, 2012, pp. 209–219.

[16] B. Nicolae, "Towards Scalable Checkpoint Restart: A Collective Inline Memory Contents Deduplication Proposal," in *IPDPS '13: The 27th IEEE International Parallel and Distributed Processing Symposium*, Boston, États-Unis, Jan. 2013. [Online]. Available: http://hal.inria.fr/hal-00781532

VIII. Government License